## CSC 180F Assignment 2

due October 13, 2000, at 9:00 a.m.; no late assignments without written explanation.

**Prime numbers**

A *factor* of an integer is a number which divides it evenly. So $d$ is a factor of $p$ exactly when $p\%d == 0$ (recall the ‘‘remainder’’ operator ‘‘%’’).

A *prime* number is a positive integer with exactly two factors: itself and 1. The numbers 2, 3, 5, and 7 are prime. 4 is not prime because it is divisible not only by 1 and 4, but also by 2. 9 is not prime because it is divisible not only by 1 and 9, but also by 3.

Prime numbers have a variety of applications; one computer application is in cryptography, especially ‘‘public key’’ cryptographic systems.

**1. Primality testing (isprime.c)**

Testing a number to see whether it is prime is a good candidate for a separate function. Its header line will be:

```
int isprime(int p)
```

It should not consult any global variables.

The argument to isprime() must be at least 2. A block comment immediately before the isprime() function should explain this additional information about the function’s interface, as well as any other comments you have.

The *isprime* function tests all numbers between 2 and $p - 1$ to see whether they divide $p$. If any of them does, the number is not prime. If none of them does, the number is prime. The function will return 1 (true) if the number is prime and 0 (false) if it is not.

Write this function and test it with the following main program (˜ajr/a2/first10.c on ecf):

```
#include <stdio.h>

int main()
{
    int i;
    extern int isprime(int p);

    printf("Primes up to 10:\n");
    for (i = 2; i <= 10; i = i + 1)
        if (isprime(i))
            printf("%d\n", i);

    return 0;
}
```

The output of this should be 2, 3, 5, 7 (on four lines and without the commas, obviously).

Your isprime() function should be in a separate isprime.c because you will link it with three further main programs below.

It should not do any input and output. That is for the calling function to do.

**2. Four-digit primes (fourdigit.c)**

The first few four-digit prime numbers are 1009, 1013, and 1019. 1007 and 1017 are not prime. Neither is 1027, 1037, or 1047. Are there any four-digit prime numbers which end in a 7? Write a program which finds any *ten* four-digit prime numbers whose base ten representation ends in a 7. Your program can assume that more than ten such numbers exist.

The last digit of the base ten representation of a number is equal to $n\%10$.

Your new file should be called fourdigit.c and it should be able to compile with your isprime.c from question one. It should output the ten numbers, only.

*(over)*

**3. Twin primes (twin.c)**

*Twin primes* are pairs of prime numbers which differ by two. (Why not one? Well, there is only one pair which is closer together than two, namely 2 and 3, because all larger even numbers are not prime on account of being divisible by two.)

3 and 5 is a twin prime pair; 5 and 7 is a twin prime pair; 107 and 109 is a twin prime pair because 107 and 109 are both prime. 105 and 107 is not a twin prime pair because 105 is not prime (even though 107 is).

Write a program using a conditional loop and your isprime() function above to find the *first two* twin prime pairs greater than 1000. It should be able to compile with your isprime.c file from question one. This ‘‘twin.c’’ program will output the four numbers representing the first two twin prime pairs greater than 1000, in order.

**4. Mersenne primes (mersenne.c)**

*Mersenne primes*, named for Marin Mersenne (1588-1648), are primes of the form $2^n - 1$ for positive integers $n$. For example, $2^2 - 1$ is 3, which is prime; hence 3 is a Mersenne prime. 5 is not a Mersenne prime even though it is indeed a prime number, because 5 is not equal to $2^n - 1$ for any positive integer $n$. 15 is not a Mersenne prime even though it is equal to $2^4 - 1$ (i.e. $2^n - 1$ for $n = 4$), because it is not prime.

The value of $2^n - 1$ gets large fairly quickly. Write a program to find the first four-digit Mersenne prime, i.e. the first Mersenne prime greater than 1000.

The best way to do this is probably as follows. Starting from $n = 1$, we see that $2^n - 1$ is 1; for $n = 2$, it’s 3; for $n = 3$, it’s 7; and so on. Eventually we reach 1000, and then we have our starting $n$. So this is a loop.

Next, *another* loop proceeding from that $n$ will find you the first appropriate $n$ such that $2^n - 1$ is prime. Your output should look like this, exactly:

```
2**13-1 is 12345, which is prime
```

(except of course that $2^{13} - 1$ isn’t 12345, and 12345 isn’t prime)

## To submit

Your program as assigned produces fairly simple output. To make sure your program is functioning as intended, you may prefer to produce other output as well, such as indications as to which numbers were checked and were *not* prime, or other intermediate results. You should be sure to *remove* any such output statements before submitting your program. In fact programmers often ‘‘comment out’’ such output statements so that they can easily be re-activated in future.

Each file must begin with a ‘‘prologue comment’’ including any useful description of the file and also your full name, with your surname in all capital letters, your student number, and your tutorial room and time (on Wednesday—the tutorial, not the lab session).

Your program should use proper indentation as illustrated by programs in lecture and in the textbook. Your files ***must*** have the names isprime.c, fourdigit.c, twin.c, and mersenne.c, and must obey the inter-file interface specifications above.

Submit your files with the command

```
submitcsc180f 2 isprime.c fourdigit.c twin.c mersenne.c
```

You may still change your files and resubmit them with the same command any time up to the due time. You can check that your assignment has been submitted with the command

```
submitcsc180f -l 2
```

You must also submit an ‘‘unsupervised term work statement’’ (on paper) in your next tutorial after the due date, using the supplied form.